VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Rychlá detekce nedostupnosti síťového připojení

Fast detection of network connection unavailability

2013 Jan Černoch

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jan Černoch**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Rychlá detekce nedostupnosti síťového připojení
Fast Detection of Network Connection Unavailability

Zásady pro vypracování:

Práce má za cíl prozkoumat existující řešení a reimplementovat vhodný mechanismus rychlé detekce nedostupnosti bezdrátového spoje v prostředí GNU/Linux. Implementovaný mechanizmus bude detekovat pravděpodobnou nedostupnost bezdrátového připojení a tzv. end-to-end spojení v časových intervalech pod jednu sekundu.

1. Seznamte se s problematikou detekce výpadků spojení v prostředí počítačových sítí a to zejména v oblasti bezdrátových sítí.
2. Navrhněte a realizujte mechanizmy pro sledování kvalitativních parametrů bezdrátových spojů (například odstup signál-šum, vysílací výkon). Za pomoci experimentu získejte informace o chování daných ukazatelů v reálném prostředí.
3. Navrhněte mechanizmus detekce výpadku na základě vyhodnocení end-to-end spojení s podporou znalosti kvalitativních vlastností probíhajícího spojení.
4. Navržené řešení implementujte a řádně otestujte. Výsledky testů zhodnoťte.

Seznam doporučené odborné literatury:

[1] GAST, Matthew. 802.11 wireless networks: the definitive guide. 2, illustrated. USA: O Reilly Media, Inc., 2005, 630 s. ISBN 9780596100520
[2] GORANSSON, Paul a Raymond GREENLAW. Secure roaming in 802.11 networks. illustrated. USA: Newnes, 2007, 343 s. ISBN 9780750682114
[3] SMITH, Roderick W. Advanced Linux networking. USA: Addison-Wesley, 2002, 752 s. ISBN 9780201774238

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Martin Milata**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2013 ..... Černoch

Abstrakt

Práce se zabývá problematikou výpadků a nedostupnosti spojení v prostředí bezdrátových sítí rodiny 802.11.

V první části této práce jsou popsány teoretické vlastnosti těchto sítí a faktory ovlivňující kvalitu spojení v nich.

Druhá část popisuje experimenty prováděné za účelem ověření teoretických předpokladů a získání dat popisujících chování těchto sítí za určitých podmínek.

Poslední část se věnuje implementaci aplikace pro operační systém GNU/Linux umožňující detekci výpadků a nedostupnosti bezdrátových sítí.

Klíčová slova: Bezdrátové sítě, 802.11, Wi-Fi, GNU/Linux

Abstract

The main aim of this bachelor thesis is network connection unavailability in wireless networks of 802.11 family.

There are described theoretical properties of these networks in the first part of this bachelor thesis. There are also described factors which affected connection quality.

The second part are about experiments which have done to verify theoretical hypotheses. There are also listed informations about behavior of these networks in specific conditions.

The last part described implementation of application to detect wireless network connection unavailability in operating system GNU/Linux.

Keywords: Wireless networks, 802.11, Wi-Fi, GNU/Linux

Seznam použitých zkratek a symbolů

AP	–	Access Point
BSSID	–	Basic Service Set Identifier
CSMA/CA	–	Carrier Sense Multiple Access with Collision Avoidance
EGID	–	Effective Group Identifier
ESSID	–	Extended Service Set Identifier
EUID	–	Effective User Identifier
GID	–	Group Identifier
GNU	–	GNU's Not Unix
ICMP	–	Internet Control Message Protocol
IP	–	Internet Protocol
MAC	–	Media Access Control
MTU	–	Maximum Transmission Unit
OS	–	Operační Systém (Operating System)
RTS/CTS	–	Request To Send/Clear To Send
SNR	–	Signal-to-Noise Ratio
TCP	–	Transmission Control Protocol
UID	–	User Identifier

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Internet se stal za posledních několik let součástí života mnoha lidí. Pro některé lidi je internet zdroj výdělku, pro jiné zdroj informací a pro další zdroj zábavy. S tím souvisí i obrovský rozmach nejrůznějších mobilních zařízení (ať už to jsou notebooky, tablety nebo chytré telefony), které se většinou neobejdou bez připojení k internetu. Ovšem připojení takových mobilních zařízení do internetu pomocí běžných kabelových sítí by bylo velice nepraktické. Díky těmto faktům se stále více dostávají do popředí bezdrátové sítě.

Tato práce se věnuje problematice výpadků a celkové nedostupnosti v prostředí bezdrátových počítačových sítí, konkrétně v prostředí Wi-Fi sítí, na operačním systému GNU/Linux.

V první části jsou popsány problémy, které se vyskytují ve Wi-Fi sítích a v bezdrátovém přenosu obecně. Jsou zde také popsány některé metody, které tyto problémy řeší.

Druhá část popisuje experimenty, které byly provedeny za účelem zjištění chování parametrů bezdrátové sítě vzhledem k detekci výpadků. Jsou zde popsány nástroje, které byly pro experimenty využity a také výsledky těchto experimentů.

Třetí část se věnuje vlastní implementaci programu na detekci výpadků v prostředí Wi-Fi sítí v OS Linux. Popisuje chování programu a implementaci jednotlivých jeho částí.

2 Teoretická část

V bezdrátových sítích se všechna data přenáší pomocí přenosového média (nejčastěji vzduchu) v podobě elektromagnetických vln. Tyto vlny spolu interferují, odráží se od různých objektů nebo prochází zdmi budov. Všechny tyto faktory mají za následek snížení propustnosti přenosu dat nebo dokonce jeho úplné znemožnění.

2.1 Síla signálu a šum

Celý vesmír je plný náhodných elektromagnetických vln označovaných jako *šum*. Hlavním faktorem určujícím kvalitu bezdrátové komunikace je, aby signál byl co nejlépe rozpoznatelný na pozadí těchto vln. Pokud se úroveň signálu přiblíží k úrovni šumu, stává se takový signál nerozeznatelný a komunikace se stává obtížnější až nemožnou [1].

Tento faktor se vyjadřuje pomocí tzv. *signal-to-noise ratio* (*SNR*) a je definován jako poměr úrovně signálu k úrovni šumu [2]:

$$SNR = \frac{P_{signal[W]}}{P_{noise[W]}} \tag{1}$$

Častěji se však vyjadřuje v decibelech [2]:

$$SNR_{dB} = 10 \log_{10} \left(\frac{P_{signal[W]}}{P_{noise[W]}} \right) = P_{signal[dB]} - P_{noise[dB]} \tag{2}$$

2.1.1 Shannon-Hartleyův teorém

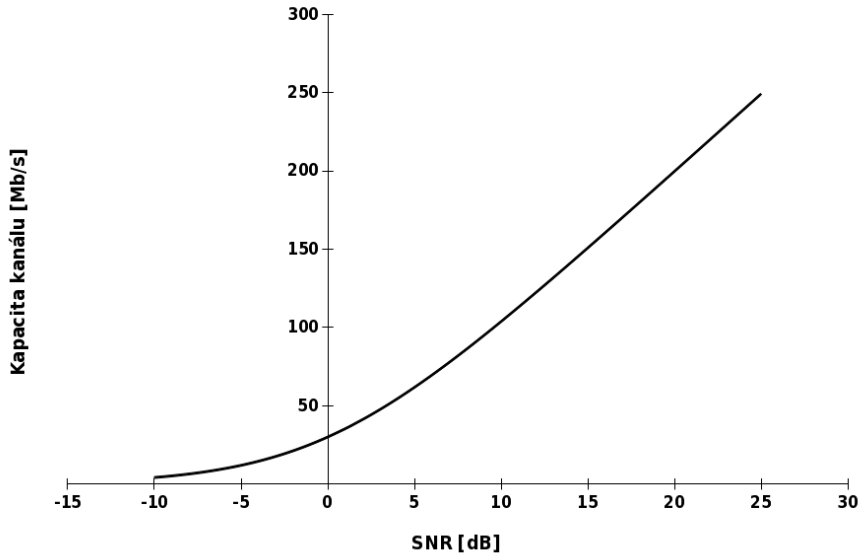
Tento teorém definuje limit (známý jako *Shannonův limit*), který určuje kolik nejvýše dat lze bezchybně přenést přes kanál s danou šířkou pásma při výskytu šumu. Pokud je tento limit překročen, začne prudce narůstat chybovost. Autory tohoto teorému jsou Claude Shannon a Ralph Hartley a byl ověřen v roce 1948 v Bellových laboratořích.

Shannon–Hartleyův teorém určuje maximální přenosovou kapacitu kanálu (*C [b/s]*) s šířkou pásma (*B [Hz]*) a průměrným odstupem signálu od šumu (*SNR*),

který musí být vyjádřen jako poměr úrovně signálu a šumu (1), nikoliv logaritmicky v decibelech (2) [1, 2]:

$$C = B \log_2 (1 + SNR) = B \log_2 \left(1 + \frac{P_{signal}}{P_{noise}}\right) \tag{3}$$

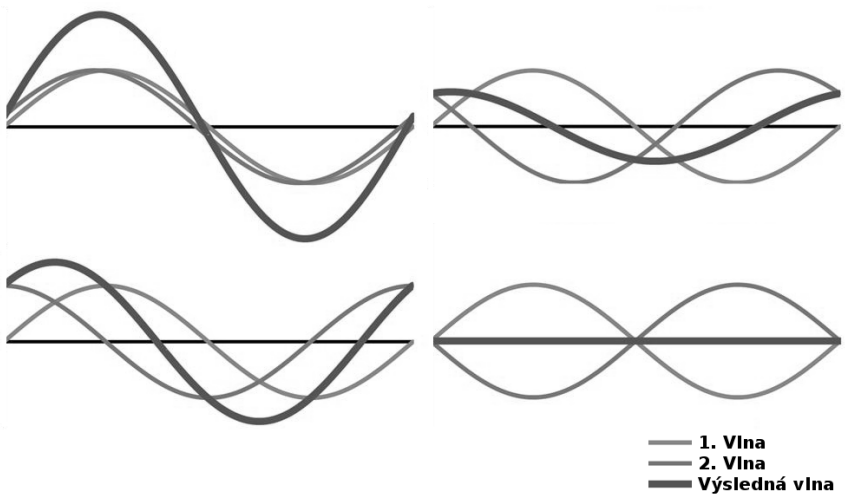
Na obrázku 1 je znázorněn graf vyjadřující Shannonův limit jako funkci SNR pro kanál, který má šířku pásma 30 MHz, což je průměrná hodnota kanálů používaných ve Wi-Fi sítích (802.11a – 20 MHz, 802.11b/g – 22 MHz, 802.11n – 40 MHz).



Obrázek 1: Shannonův limit jako funkce SNR pro kanál se šířkou pásma 30 MHz

2.2 Vlnová interference

Elektromagnetické vlny se navzájem ovlivňují. Pokud dvě či více vln dorazí k přijímači ve stejný okamžik, výsledná vlna je součtem těchto vln. Tento jev nejlépe ilustruje obrázek 2. Jak lze vidět, může nastat i situace, že se dvě vlny (pokud mají opačnou fázi) navzájem zcela vyruší.



Obrázek 2: Vlnová interference [4]

Interference může nastat v několika případech:

- **Více vysílačů vysílá ve stejný okamžik** – Pokud více vysílačů začne vysílat svá data ve stejný okamžik, může dojít ke vzájemnému ovlivnění. Tento problém je nazýván jako *kolize*.
- **Zpoždění z důvodu odrazů** – Většina vysílačů je všesměrových, proto může nastat situace, kdy jedna vlna dorazí k přijímači vícekrát. Protože každá vlna putuje jinou cestou, některé přímo od vysílače k přijímači, ostatní se po cestě různě odráží, dorazí k přijímači vůči sobě v různý čas, což může způsobit vzájemnou interferenci.
- **Interference s jinými druhy vln** – Některé přístroje vyzařují elektromagnetické vlny (u Wi-Fi sítí v pásmu 2,4 GHz jsou to např. mikrovlnné trouby, v pásmu 5 GHz pak zase některé radary a sateliní systémy), které mohou ovlivňovat rádiový přenos [1].

2.3 Další faktory ovlivňující bezdrátovou komunikaci

- **Vzdálenost vysílače a přijímače** – Čím větší vzdálenost elektromagnetické vlny urazí, tím více klesá úroveň signálu a ten se tak dostává blíže k úrovni šumu (SNR se zmenšuje).

- **Přenosová rychlost** – Vyšší přenosová rychlost znamená odeslání více bitů za stejný časový interval. Aby byl přijímač schopen rozeznat jednotlivé bity, je potřeba čistší signál (větší SNR).
- **Velikost přenášených dat** – Čím větší velikost mají přenášená data, tím delší čas putují vzduchem a je tak větší riziko interference s jinými vlnami [1].

2.4 Problémy typické pro Wi-Fi sítě

2.4.1 Ztráta rámce

Ve Wi-Fi sítích se data posílají rozdělená v podobě tzv. *rámců* a využívá se pozitivního potvrzování, což znamená, že pro každý přijatý rámec musí být odesláno potvrzení, že byl přijat. Pokud však vysílací stanice nedostane na odeslaný rámec žádné potvrzení (ať už z důvodu že se ztratil samotný rámec nebo jeho potvrzení), je rámec považován za ztracený a většinou musí být poslán znova [1].

2.4.2 Problém skrytého uzlu

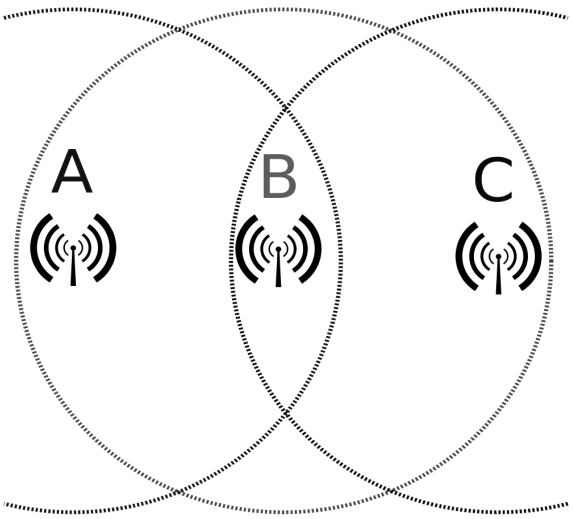
Wi-Fi sítě jsou konstruovány tak, že pokud nějaká stanice vysílá, ostatní musí pouze poslouchat, jinak dochází ke kolizím. Pokud chce stanice vysílat, musí počkat, až nebude vysílat žádná jiná stanice (protokol *CSMA/CA*). Problém může nastat v případě rozlehlejších sítí, kdy jedna stanice je sice v dosahu Access Pointu, ale nemusí být v dosahu některých dalších stanic a nedokáže detekovat, zda tyto stanice vysílají nebo ne. Na obrázku 3 je tento problém znázorněn.

Stanice A a C jsou vůči sobě skryté a ani jedna z nich nedokáže detekovat vysílání té druhé. Může tedy nastat situace, že začnou vysílat obě dvě najednou a dojde ke kolizi [1].

2.5 Řešení používaná ve Wi-Fi sítích

2.5.1 Proměnlivá přenosová rychlost

Stanice umožňují na základě svých algoritmů implementovaných v ovladačích bezdrátových síťových karet dynamicky měnit přenosovou rychlost v reakci na měnící se podmínky. Každá stanice musí umět rozpoznat okamžik (sledováním



Obrázek 3: Problém skrytého uzlu [5]

SNR a úspěšnosti předchozí komunikace), kdy svou přenosovou rychlost musí snížit, a naopak, kdy je vhodné ji opět zvýšit. Změna přenosové rychlosti neprobíhá plynule, ale skokově, nejčastěji na nějaký násobek aktuální rychlosti. S tímto souvisí i fakt, že Access Point musí být schopen komunikace na několika přenosových rychlostech zároveň [1].

2.5.2 Fragmentace velkých rámců

Ke zvýšení propustnosti sítě lze také na stanicích nastavit tzv. *Fragmentation threshold*. Jedná se o hodnotu vyjadřující maximální velikost rámce. Rámce větší než tato hodnota budou rozděleny na několik menších rámců. Případné chyby se tak budou týkat pouze malých rámců, které mohou být znova poslány rychleji než rámce velké. Na druhou stranu, příliš malá hodnota má za následek snížení propustnosti sítě, protože každý fragment musí být potvrzen (kapitola 2.4.1) [1].

2.5.3 Protokol RTS/CTS

Protokol *RTS/CTS* odstraňuje problém skrytých uzlů (kapitola 2.4.2). Pokud stanice chce vysílat, pošle nejprve cílové stanici (v případě infrakstrukturních sítí je to AP, který je v dosahu všech ostatních stanic) RTS rámec s požadovaným časem potřebným ke komunikaci, čímž si zarezervuje přístup k médiu. Cílová stanice

(nebo AP) poté odešle zpět rámec CTS s časem, po který může stanice vysílat. Všechny stanice, které zachytí CTS rámec (v případě infrastrukturní sítě jsou to všechny stanice asociované s AP, který odeslal CTS rámec), se poté musí na tuto dobu odmlčet.

Nevýhoda tohoto protokolu ovšem tkví v tom, že značně snižuje propustnost sítě. Důvodem je zpoždění před vlastním odesláním rámce, kvůli nutné výměně RTS a CTS rámců. V praxi se proto tento protokol využívá pouze u rozlehlých (zejména venkovních) sítí, kde je větší riziko výskytu skrytých uzlů.

Určitým kompromisem je tzv. *RTS threshold*. Tato hodnota se nastavuje na jednotlivých stanicích a určuje, že rámce větší než tato hodnota se budou posílat pomocí protokolu RTS/CTS, zatímco pro menší rámce bude použit protokol CSMA/CA [1].

2.5.4 Kontrola vysílacího výkonu

Toto řešení se týká pouze Wi-Fi sítí v pásmu 5 GHz. Stanice umožňují regulovat svůj vysílací výkon na nejnižší možnou úroveň, ovšem tak, aby nedocházelo ke ztrátě kvality bezdrátového spojení. Tímto mechanismem omezují svůj dosah pouze na nezbytnou vzdálenost a tím omezují i případné interference s ostatními stanicemi a přístroji (kapitola 2.2) [1].

2.5.5 Roaming

Tzv. *roaming* je záležitostí pouze sítí s více propojenými AP. Jedná se o plynulé přecházení stanice mezi jednotlivými AP. Rozhodnutí, kdy se asociovat s jiným AP závisí na ovladači bezdrátové síťové karty a na jeho výrobci. Některé karty sledují kvalitu aktuálního spojení, a pokud začne být nízké SNR a přenosová rychlost, začnou hledat vhodnější AP. Jiné naopak sledují ostatní AP průběžně a připojují se k tomu s nejsilnějším signálem. A některé síťové karty se přepojí až když je signál zcela ztracen [1].

3 Experimentální část

Cílem experimentů bylo zjistit, jaké všechny informace týkající se bezdrátového síťového spojení je možné získat od linuxového jádra, resp. ovladačů bezdrátových síťových karet. Dále bylo cílem zjistit, jak se tyto údaje mění v reálném prostředí a jak pomocí nich detekovat výpadky a celkovou nedostupnost bezdrátového spojení.

3.1 Nástroje OS Linux

Pro operační systém Linux existuje mnoho nástrojů pro sledování parametrů počítačových sítí včetně těch bezdrátových.

3.1.1 Program `iw`

Tento nástroj se využívá k zobrazování, manipulaci a konfiguraci bezdrátových síťových rozhraní. Lze se pomocí něj např. připojovat k bezdrátovým sítím, vyhledávat dostupné sítě, vytvářet nové sítě a v neposlední řadě také zobrazovat parametry o konkrétní bezdrátové síti.

`iw dev wlan0 station dump`

Tento příkaz slouží k výpisu informací o bezdrátové síti, ke které je zadané síťové rozhraní aktuálně připojeno. Nejdůležitější informace, které tento příkaz poskytuje se nachází ve výpisu 1.

```
Station c8:60:00:72:50:60 (on wlan0)
    rx bytes:       1329047051
    rx packets:     909949
    tx bytes:       30374822
    tx packets:     312664
    tx retries:     323
    tx failed:      0
    signal:         −27 dBm
    tx bitrate:     72.2 MBit/s MCS 7 short GI
```

Výpis 1: Výpis programu `iw`

Na prvním řádku se nachází *BSSID* (MAC adresa AP) bezdrátové sítě a název síťového rozhraní. Poté následuje statistika přijatých a odeslaných bytů a paketů. Za touto statistikou následuje počet rámců (*tx retries*), které bylo potřeba z důvodu ztráty odeslat znova. Pokud počet neúspěšných odeslání jednoho rámce dosáhne určité úrovně, není tento rámec už dále posílán a je započítán do položky *tx failed*. Na konci výpisu je uvedena ještě úroveň signálu a aktuální přenosová rychlost.

`iw dev wlan0 survey dump`

Tento příkaz vypisuje informace o všech kanálech Wi-Fi sítě, zobrazuje jejich frekvenci, úroveň šumu a další informace.

Poznámka 3.1 Bohužel ne všechny ovladače bezdrátových síťových karet tyto informace poskytují. Např. ovladače pro bezdrátové síťové karty výrobce *Atheros* tyto informace poskytují, zatímco ovladače od firmy *Intel* ne. Z pozorování však (díky bezdrátové síťové kartě od firmy Atheros) vyplynulo, že úroveň šumu ve Wi-Fi sítích je velice podobná a pohybuje se v okolí hodnoty *-90 dBm*. Z tohoto důvodu se veškerá detekce výpadků a nedostupnosti bude vyhodnocovat pouze podle úrovně signálu, nikoliv podle SNR. Avšak tato hodnota nebude hlavním ukazatelem výpadku.

3.1.2 Program `ip`

Dalším nástrojem používaným v experimentech byl nástroj `ip`. Tento nástroj se nespecializuje pouze na bezdrátová síťová rozhraní, ale lze pomocí něj manipulovat a zobrazovat informace o všech síťových rozhraních v systému. Lze pomocí něj např. aktivovat nebo deaktivovat jednotlivá síťová rozhraní, přiřazovat jim IP adresy, nastavovat výchozí bránu, aj.

`ip -s link show dev wlan0`

Tento příkaz zobrazuje statistiku provozu na zadaném rozhraní (výpis 2).

2: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DORMANT qlen 1000 link/ether 00:16:ea:60:f6:a4 brd ff:ff:ff:ff:ff:ff

```
RX: bytes  packets  errors  dropped overrun mcast
1312018915 885672 0       0       0       0
TX: bytes  packets  errors  dropped carrier collsns
30939870  315727  0       0       0       0
```

Výpis 2: Výpis programu `ip`

První dva řádky opět zobrazují informace o síťovém rozhraní (název, MTU, zda-li je aktivní, MAC adresu, atd.). Ostatní řádky obsahují samotnou statistiku, nejprve pro příchozí provoz (*RX*) a poté pro provoz odchozí (*TX*). Nejdůležitějšími hodnotami jsou počet bytů (*bytes*), paketů (*packets*), počet chybných paketů (*errors*) a počet zahozených paketů (*dropped*), což jsou pakety, které musely být zahozeny, např. kvůli tomu, že chyběl jeden z jejich fragmentů.

3.1.3 Soubor `/proc/net/wireless`

V adresáři `/proc/net` se nachází soubor `wireless`, který také poskytuje informace o bezdrátové síti. Kromě údajů dostupných přes program `iw`, jako je úroveň signálu, šumu (poznámka 3.1), aj., se zde nachází ještě několik dalších údajů (výpis 3).

```
Inter- | sta- | Quality        | Discarded packets               | Missed | WE
 face  | tus  | link level noise | nwid crypt  frag  retry  misc | beacon | 22
wlan0:  0000   70.  -28.  -256     0     0     0     554    3        0
```

Výpis 3: Výpis souboru `/proc/net/wireless`

Prvním z nich je *Link Quality*. Jedná se o subjektivní hodnocení kvality bezdrátového spojení poskytované ovladačem bezdrátové karty. Dalšími parametry jsou *frag*, *retry* a *misc* v sekci *Discarded packets*. První jmenovaný určuje kolik přijatých paketů nebylo možno sestavit, např. proto, že chyběl některý z rámců. Parametr *retry* odpovídá hodnotě *tx failed* z výpisu programu `iw` a parametr *misc* určuje pakety, které byly ztraceny z jiných důvodů.

Posledním parametrem je parametr *Missed beacon*, který určuje, jak již název napovídá, kolik tzv. *beacon rámců* nebylo stanicí zachyceno. Beacon rámce jsou rámce vysílané AP v pravidelných intervalech a nesoucí informace o aktuálním stavu sítě. Každá stanice musí naslouchat těmto rámcům.

Poznámka 3.2 Protože u parametru *Link Quality* se nejedná o objektivní hodnocení kvality spojení, ale je závislé na implementaci výrobce bezdrátové síťové karty, nebude proto tento parametr používán k detekci výpadků.

3.1.4 Program `ping`

Posledním programem, který byl použit v experimentech byl program `ping` sloužící k měření latence sítě. Využívá k tomu protokol *ICMP*, který je primárně určen na oznamování zpráv v počítačových sítích (nejčastěji chyb, např. o nedostupnosti cílové stanice, atp.).

Program odešle paket s žádostí (*Echo request*) na cílovou stanici a měří čas, za jak dlouho dorazí odpověď (*Echo reply*). Tento program umožňuje nastavit těmto ICMP paketům různé vlastnosti, ať už je to jejich velikost, počet, interval v jakých se budou posílat nebo maximální doba čekání na odpověď.

3.2 Testovací sítě

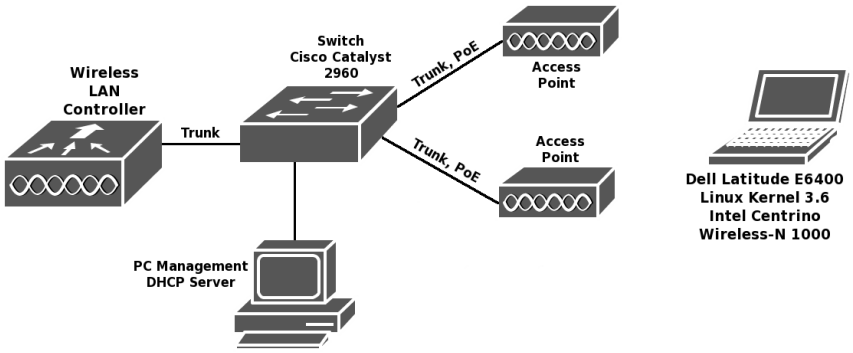
Experimenty byly prováděny na čtyřech bezdrátových sítích.

1. **Univerzitní síť VŠB-TU Ostrava „tuonet-peap“** – Jedná se o síť s mnoha AP a značným provozem. V této síti tedy bylo možno otestovat, jak se parametry sítě v takovém provozu chovají a jak se chovají při roamingu (kapitola 2.5.5).

2. **Experimentální síť** – Jedná se o Wi-Fi síť zkonstruovanou konkrétně za účelem těchto experimentů. Její schéma je znázorněno na obrázku 4.

3. **Wi-Fi síť č. 1** – Jedná se o klasickou topologii infrastrukturní sítě s jedním AP umístěnou v prostředí, kde se nachází i několik dalších bezdrátových sítí (některé i na stejném kanálu).

4. **Wi-Fi síť č. 2** – Jedná se o stejnou síť jako *Wi-Fi síť č. 1*, ovšem umístěnou mimo dosah ostatních bezdrátových sítí.



Obrázek 4: Schéma experimentální sítě

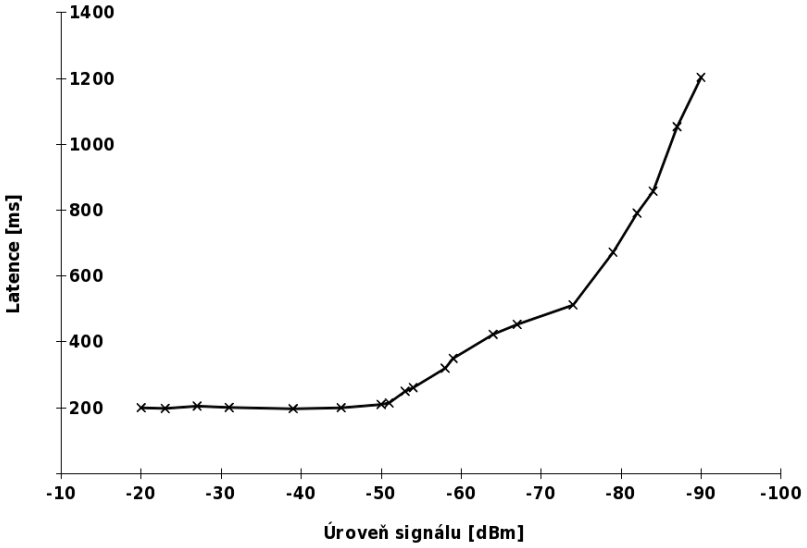
3.3 Průběh experimentů

Průběh experimentů byl vždy stejný. Na stanici se sbíraly pomocí výše zmíněných nástrojů (kapitola 3.1) informace o síti, zatímco se tato stanice pohybovala v síti mezi jednotlivými AP (pouze v sítích s více AP) až se postupně dostala za hranici dané sítě. Mezitím další stanice v síti generovaly provoz. Testovací stanice neměla nastavený *Fragmentation threshold* ani *RTS threshold* (kapitoly 2.5.2 a 2.5.3).

3.4 Výsledky experimentů

V experimentech se projevily všechny teoretické předpoklady (kapitola 2).

- Se zvětšující se vzdáleností klesá úroveň signálu. Vzhledem k poznámce 3.1 bylo zjištěno, že pokud úroveň signálu klesne k hodnotě *-80 dBm*, začíná prudce růst latence sítě a ta se postupně stává nepoužitelnou (obrázek 5).
- Latenci sítě také ovlivňuje její celková zatíženost. V sítích s velkým provozem byla latence při stejné přenosové rychlosti řádově vyšší než v síti s minimálním provozem.
- Větší rámce jsou náchylnější k rušení, než rámce menší. Větší ICMP pakety se začaly ztrácet již při vyšší úrovni signálu než pakety menší.
- Pokud se začnou objevovat nějaké chybějící *beacon rámce* (kapitola 3.1.3), v naprosté většině případů to znamená, že se stanice dostala mimo dosah



Obrázek 5: Závislost latence sítě na úrovni signálu

sítě. Prakticky vždy, jakmile se tyto chybějící rámce začaly objevovat, zasáhlo linuxové jádro, resp. ovladače síťových karet, a stanice se od sítě odpojila.

- Pokud se některý z parametrů *frag*, *retry* nebo *misc* (kapitola 3.1.3) začne objevovat ve velké míře (cca 75 % všech odeslaných rámců, příp. pro parametr *frag* rámců přijatých), během několika okamžiků následuje odpojení od sítě. Ve většině případů se však dříve než tato situace nastane objeví jiný problém vedoucí k nepoužitelnosti sítě (např. příliš vysoká latence).

- Ovladače bezdrátových karet skutečně v závislosti na okolních podmínkách skokově mění svou přenosovou rychlost.

- V sítích s více AP spolehlivě fungoval *roaming*. Jen ve velice ojedinělých případech došlo k situaci, že bezdrátové spojení nebylo dostupné, i když stanice byla v dosahu AP. Tyto situace mohly být způsobeny např. rychlým pohybem stanice.

Bohužel se v průběhu experimentů ani v jednom případě nepodařilo nasimulovat prostředí tak, aby ve výpisu programu `ip` (výpis 2) vznikaly chyby nebo některé pakety musely být zahazovány.

4 Implementační část

Vlastní program je napsán v programovacím jazyce *Python* [7] (konkrétně ve verzi 3.3). Jedná se o open source dynamický objektově orientovaný[1] interpretovaný programovací jazyk. Bývá často považován za jazyk vhodný pouze k učení, ale jeho možnosti jsou daleko větší – od jednoduchých skriptů až po rozsáhlé programy s uživatelským rozhraním, aplikace s přístupem k databázovým serverům nebo internetové aplikace.

Důvodů, proč byl zvolen právě tento jazyk, je několik:

- **Jednoduchost** – Ovšem ne z hlediska funčnosti, ale jednoduchosti a čistoty syntaxe.
- **Rychlost vývoje** – Syntaxe jazyka je jednoduchá, stručná a velmi dobře čitelná, proto i vývoj v něm je rychlejší než v některých jiných jazycích.
- **Množství knihovních funkcí a modulů** – Standardní knihovna Pythonu obsahuje nepřeberné množství již připravených funkcí a modulů (např. moduly pro práci s konfiguračními soubory, pro práci s různými internetovými protokoly, aj.). Další moduly je možné získat na internetu jako „moduly třetích stran“ (např. PyQt pro práci s knihovnou Qt).
- **Provázanost s OS Linux** – Přesto, že je Python multiplatformní jazyk, velmi dobře spolupracuje s operačním systémem Linux. Obsahuje moduly pro systémová volání (`ioctl()`), funkce pro práci s procesy (`fork()`, ...), atd. Mnoho aplikací pro OS Linux je napsáno právě v tomto jazyce.
- **Spolupráce s jazyky C/C++** – Python velmi dobře spolupracuje s jazyky C/C++. Umí pracovat nejen s datovými typy jazyků C/C++, ale i s jejich poli, strukturami a zvládne také načítat a spouštět funkce z dynamických knihoven.

[1] Podporuje ale i procedurální a částečně funkcionální paradigmata.

4.1 Verze programu

4.1.1 První verze

První verze programu byla napsána jako jednoduchý skript. Skript byl použit v experimentech pro automatické získávání aktuálních parametrů Wi-Fi sítě. V nekonečné smyčce se v podprocesech spouštěly výše zmiňované nástroje OS Linux (kapitola 3.1) a z výstupů těchto programů získaval skript potřebné parametry.

4.1.2 Druhá verze

Hlavními nedostatky první verze programu byly:

- Režie spojená s opakovaným spouštěním stále nových podprocesů.
- Velká časová náročnost daná zpracováváním textových výstupů programů volaných v podprocesech.

Kvůli těmto nedostatkům byl program od základu přepsán. Všechny informace o parametrech sítě lze získat i přímo od linuxového jádra, popř. od ovladačů bezdrátových síťových karet.

V operačním systému Linux existují dva speciální adresáře. Jedná se o adresář `/proc` a adresář `/sys`.

Adresář `/proc`

Do tohoto adresáře je připojen virtuální souborový systém *proc*. Nejedná se však o souborový systém v pravém slova smyslu, fyzicky neexistuje na žádném disku, nachází se pouze v paměti počítače. Umožňuje získávat aktuální informace o jádře systému a jednotlivých procesech, příp. některé parametry i měnit.

Práce s ním je stejná jako s většinou věcí v Linuxu – vše je soubor. Ovšem soubory v něm fyzicky neexistují, při čtení je jádro automaticky generuje, při zápisu se parametry ihned nastavují [6].

Adresář `/sys`

Tento adresář je obdobou `/proc`. Je do něj připojen virtuální souborový systém *sysfs*, ale zatímco v `/proc` jsou uchovávány informace o jádře systému a procesech, adresář `/sys` uchovává informace o zařízeních připojených k počítači (síťové karty, pevné disky, grafické karty, atd.) [6].

Vzhledem k tomu, že např. síťová rozhraní nejsou v Linuxu jako jedna z mála věcí reprezentována jako soubor, ne všechny informace lze získat z výše zmíněných adresářů. Operační systém Linux poskytuje ještě další možnost, jak získat požadované informace – systémová volání `ioctl()`.

Systémová volání `ioctl()`

Jedná se o speciální funkci určenou pro komunikaci mezi tzv. user space procesy (procesy běžícími v neprivilegovaném režimu) a samotným jádrem operačního systému, resp. ovladači jednotlivých zařízení.

Funkce přebírá tři argumenty: tzv. file descriptor (tj. identifikátor, pomocí kterého se přistupuje k otevřenému souboru – v případě práce s počítačovou sítí se ve většině případů jedná o otevřený socket), číselnou konstantu určující požadovanou operaci (každé zařízení má vlastní ioctl operace definované v hlavičkových souborech) a parametr [6].

4.2 Vlastnosti programu

Program je koncipován jako démon běžící na pozadí. Po spuštění program zjistí jestli je systém připojen k nějaké Wi-Fi síti.

Pokud není systém připojen k žádné bezdrátové síti, program v pravidelných intervalech (jednou za 5 sekund[2]) kontroluje bezdrátová síťová rozhraní dostupná v systému (lze změnit pomocí konfiguračního souboru), zda-li se nepřipojilo k nějaké síti. Pokud ano, přechází program do další fáze.

[2]Častěji není potřeba, protože asociace počítače s AP je časově náročná operace.

Po připojení počítače k bezdrátové síti program cyklicky zjišťuje aktuální parametry bezdrátové sítě, ke které je systém připojen, měří latenci sítě, kontroluje end-to-end spojení a snaží se detekovat výpadky.

V případě detekování výpadku je v podprocesu spuštěna uživatelsky definovaná akce (příkaz, skript nebo program) a podle nastavení v konfiguračním souboru se buď démon ukončí anebo pokračuje znova od začátku.

4.3 Popis implementace

4.3.1 Parametry příkazové řádky a konfigurační soubor

Ovlivnit běh programu lze dvěma způsoby – zadáním parametrů při spouštění programu z příkazové řádky a pomocí konfiguračního souboru.

Pro zpracovávání parametrů příkazové řádky poskytuje standardní knihovna jazyka Python modul `argparse` [8]. Práce s tímto modulem je velice jednoduchá, jak lze vidět ve výpisu 4.

```
parser = ArgumentParser()
parser.add_argument("-d", "--debug", action="store_true", help="show debug
    messages")
args = parser.parse_args()
debug_enabled = args.debug
```

Výpis 4: Ukázka práce s `argparse`

Základem je vytvoření instance třídy `ArgumentParser`. Té se poté přidají parametry (pomocí metody `add_argument()`), které má program zpracovávat. Nakonec se zavolá metoda `parse_args()`, která vytvoří objekt s atributy odpovídajícími jednotlivým parametrům.

Výhoda toho modulu je, že automaticky generuje přepínače `-h` a `--help`, které zobrazí nápovědu pro všechny zadané parametry (argument *help* předávaný metodě `add_argument()` ve výpisu 4).

Standardní knihovna jazyka Python také poskytuje modul pro zpracovávání konfiguračních souborů. Název modulu je `configparser` [10] a lze s jeho po-

mocí nejen jednoduše zpracovávat již vytvořené, ale také vytvářet nové konfigurační soubory.

Konfigurační soubor musí obsahovat alespoň jednu sekci (ve tvaru: `[sekce]`). Každá sekce obsahuje vlastní volby (ve tvaru: `volba = hodnota`). Ukázka práce s tímto modulem je ve výpisu 5.

```
parser = ConfigParser()
parser.read(/path/to/configuration/file )
option = parser[section_name][option_name]
```

Výpis 5: Ukázka práce s `configparser`

Základem je opět instance třídy, tentokrát `ConfigParser`. Poté je na tento objekt zavolána metoda `read()`, která přebírá cestu ke konfiguračnímu souboru. Konfigurační soubor pak je reprezentován jako slovník sekcí, z nichž každá sekce je slovníkem voleb.

Seznam všech přípustných parametrů příkazové řádky a voleb v konfiguračním souboru je uveden v příloze C.

4.3.2 Detekce připojení k Wi-Fi síti

Program umí detekovat, zda-li je počítač připojen k bezdrátové síti nebo ne (toto chování je implementováno ve třídě `ConnectDetector`). Detekce je složena z několika kroků. Všechny kroky (metody třídy) se cyklicky provádějí dokud se všechny úspěšně nedokončí. Pokud kterýkoliv z nich selže, začíná se od začátku.

Detekce bezdrátových rozhraní v systému

Tento krok je závislý na nastavení v konfiguračním souboru. Pokud je nastavena volba `interface` na určité rozhraní, bude se vyhledávat pouze to. V ostatních případech se budou hledat všechna bezdrátová síťová rozhraní v systému.

Informace o všech síťových rozhraních dostupných v systému jsou uloženy v adresáři `/sys/class/net`. Adresáře jednotlivých rozhraní jsou pojmenované podle daného rozhraní. Pro zjištění všech síťových rozhraní tedy stačí zjistit obsah tohoto adresáře.

Zda-li je rozhraní bezdrátové nebo ne závisí na tom, existuje-li v jeho adresáři (tedy např. `/sys/class/net/wlan0`) soubor `wireless`.

V dalších krocích programu se pracuje již pouze s bezdrátovými rozhraními.

Detekce aktivace rozhraní a připojení k síti

V adresáři každého jednotlivého rozhraní se nachází dva soubory, `operstate` a `carrier`. V prvním jmenovaném souboru se nachází stav rozhraní (tj. *up* nebo *down*). Ve druhém souboru je uložena hodnota, která určuje, zda je připojen „nosič“ dat (u drátových rozhraní, zda-li je připojen síťový kabel, u bezdrátových rozhraní, zda-li je počítač asociován s nějakým AP). Pokud rozhraní není aktivní (tj. v souboru `operstate` je hodnota *down*), způsobí pokus o čtení ze souboru `carrier` chybu *Invalid Argument*. V ostatních případech je v souboru `carrier` uložena hodnota *0* nebo *1*.

Další kroky programu pracují pouze s rozhraním, které bylo jako první vyhodnoceno jako připojené.

Informace o bezdrátové síti

Pokud je nějaké rozhraní vyhodnoceno jako připojené, program se pokusí zjistit základní informace o bezdrátové síti, tj. *ESSID* a *BSSID*. Obě dvě hodnoty je třeba získat pomocí `ioctl()` volání, jak je ukázáno ve výpisu 6.

```
SIOCGIWAP = 0x8B15

iwreq = struct.pack("16sH14s", b"wlan0", socket.AF_INET, bytes(0))

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
bssid = fcntl.ioctl(sock.fileno(), SIOCGIWAP, iwreq)
sock.close()

bssid = struct.unpack("16sH14s", bssid)[-1]
```

Výpis 6: Získání aktuálního BSSID pomocí `ioctl()`

V hlavičkovém souboru `linux/wireless.h` je definovaná číselná konstanta `SIOCGIWAP`, která slouží pro získání BSSID. Ve stejném hlavičkovém souboru

je také definovaná struktura `iwreq`, která je předávána při systémovém volání jako parametr. První položkou ve struktuře `iwreq` je jméno síťového rozhraní (převedeno na sekvenci bytů), další určuje typ socketu, který bude předáván při systémovém volání jako file descriptor (AF_INET určuje IPv4 socket). Protože se data získávají, parametr není potřeba, proto se použijí nuly.

Při `ioctl()` volání v Pythonu se parametr této metody předává jako sekvence bytů odpovídající struktuře jazyků C/C++. O převedení proměnných Pythonu na sekvenci bytů se stará metoda `pack()` z modulu `struct` [14]. Prvním argumentem této metody je tzv. formátovací řetězec určující kolik argumentů se bude převádět a na jaký formát (*16sH14s* znamená, že druhý argument bude převeden na pole znaků o velikosti 16, třetí argument bude převeden na unsigned short integer a čtvrtý na pole znaků o velikost 14).

Dále se vytvoří socket, který se použije jako file descriptor a zavolá se metoda `ioctl()` z modulu `fcntl` [11]. Ta vrátí sekvenci bytů odpovídající struktuře `iwreq`, jejíž poslední položka obsahuje BSSID.

Převod sekvence bytů zpět na proměnné Pythonu provede metoda `unpack()` z modulu `struct`, která podle formátovacího řetězce vrací odpovídající n-tici (BSSID se nachází na poslední pozici n-tice).

ESSID se zjišťuje obdobně pomocí systémového volání a `SIOCGIWESSID` konstanty.

Dalším parametrem, který se o síti zjišťuje je MTU. Hodnota v bytech je uložena v souboru `mtu` v adresáři síťového rozhraní.

IP adresa rozhraní

Dalším krokem v pořadí je získání IP adresy přiřazené síťovému rozhraní. To je provedeno taktéž pomocí systémového volání s konstantou `SIOCGIFADDR`.

IP adresa výchozí brány

Směrovací tabulka je uložena v souboru `/proc/net/route` (výpis 7).

```
Iface   Destination   Gateway   ...
```

```
wlan0   00000000    0100000A  ...
wlan0   0000000A    00000000  ...
```

Výpis 7: Výpis části směrovací tabulky

IP adresy v tomto souboru jsou uloženy jako 8 šestnáctkových číslic, kdy každé dvě číslice tvoří jeden oktet IP adresy. Oktety jsou zapsány v opačném pořadí.

V prvním řádku je vždy uložena implicitní cesta. Další řádky vyjadřují jednotlivá rozhraní a sítě. Brána každé sítě je uvedena ve sloupci *Gateway*. Pokud je v tomto sloupci uvedena adresa *0.0.0.0*, znamená to, že brána je zároveň implicitní cestou a použije se hodnota z prvního řádku (ve výpisu 7 je adresa výchozí brány *10.0.0.1*).

Ping na výchozí bránu

Pokud se všechny výše uvedené kroky dokončí bez chyby, je spojení otestováno pomocí ICMP paketu zaslaného na adresu výchozí brány (kapitola 4.3.3).

Test end-to-end spojení

Posledním krokem je otestování tzv. end-to-end spojení (kapitola 4.3.4).

Pokud se všechny předchozí kroky provedly úspěšně, je počítač připojen k bezdrátové síti a je zahájena fáze detekce výpadků.

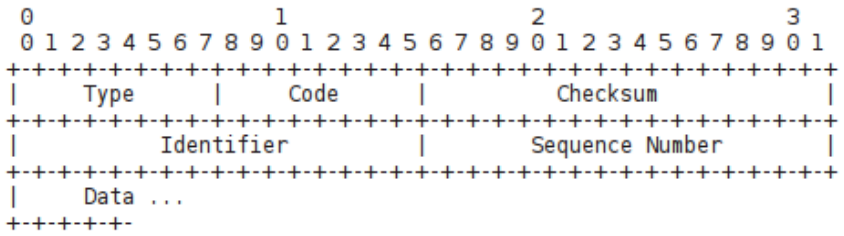
4.3.3 Implementace programu Ping

Program také obsahuje vlastní implementaci programu `ping` (kapitola 3.1.4). To s sebou ovšem nese jednu nevýhodu v podobě nutnosti spouštět program se superuživatelskými právy (4.3.6).

Tento program je v programu implementován jako samostatná třída a umožňuje vytvářet ICMP pakety různých velikostí, umožuje nastavit maximální čas na odpověď (*timeout*) a umožňuje posílat ICMP pakety dávkově (tj. odeslání více paketů ihned za sebou).

Popis ICMP paketu

ICMP paketů je několik druhů. Přenáší se jako data v IP paketu (tzv. *enkapsulace*). Pro `ping` se používají již výše zmíněné tyto dva druhy paketu: *Echo request* a *Echo reply*. Většina ostatních ICMP paketů (např. *Destination unreachable* nebo *Time exceeded*) slouží k oznamování chyb. Každý ICMP paket je tvořen hlavičkou a daty. Schéma ICMP paketů „echo“ paketů je na obrázku 6.



Obrázek 6: Schéma paketů *echo request* a *echo reply* [15]

Hlavička má velikost 8 bytů. Prvních 16 bitů obsahuje dvě 8bitové hodnoty pro typ (*8* označuje echo request a *0* echo reply) a kód (pro oba dva pakety je kód *0*). Dalších 16 bitů je vyhrazeno pro kontrolní součet (*checksum*). Následují dvě 16bitové hodnoty označující identifikátor a pořadové číslo (tyto hodnoty musí být v žádosti i odpovědi stejné).

Za hlavičkou se nacházejí libovolná data libovolné délky[3] (která musí být taktéž v žádosti i odpovědi stejná) [15].

Vytváření ICMP paketů

Vytvoření ICMP paketu se sestává z několika částí. Nejprve je potřeba vygenerovat hlavičku a data (výpis 8).

```
header = struct.pack("!2B3H", 8, 0, 0, id, seq)
data = bytes([i & 0xFF) for i in range(0, size)])
packet = header + data
```

Výpis 8: Vytvoření ICMP paketu

Hlavička paketu je vygenerována pomocí výše zmiňovaného modulu `struct`. Formátovací řetězec odpovídá struktuře hlavičky paketu (obrázek 6). Znak „!“

[3] Nejlépe ovšem takové délky, aby nedošlo k fragmentaci IP paketu.

ve formátovacím řetězci znamená, že argumenty budou převedeny do síťového pořadí bytů (*big-endian*). Typ paketu je nastaven na *8* a kód na *0*. Kontrolní součet se prozatím nastaví na *0* [15]. Identifikátor je pro každou instanci třídy `Ping` náhodně vygenerován. Pořadové číslo se po každém vytvoření paketu inkrementuje o 1. Poté jsou vygenerována data o určité velikosti v podobě opakujících se posloupností bytů s hodnotami 0–255.

Nakonec je potřeba spočítat kontrolní součet (výpis 9) a vložit jej do hlavičky paketu.

```
if len(packet) % 2 == 1:
    packet += b"\x00"

words = array.array("H", packet)

checksum = sum(words)

while checksum > 0xFFFF:
    checksum = (checksum & 0xFFFF) + (checksum >> 16)

checksum ^= 0xFFFF

checksum = socket.htons(checksum)
```

Výpis 9: Výpočet kontrolního součtu ICMP paketu

Pokud počet bytů paketu je lichý, je nakonec paketu přidán jeden nulový byte. Poté je paket pomocí modulu `array` [9] rozdělen na 16bitová slova, která jsou posléze sečtena. Dalším krokem je sečtení prvních 16 bitů tohoto součtu se zbylými bity. Tento krok je opakován tak dlouho, dokud výsledek nelze uložit pomocí nejvýše 16 bitů. Nakonec je kontrolní součet znegován a převeden do síťového pořadí bytů [16].

Nejjednodušší způsob jak vložit kontrolní součet do hlavičky je vytvořit novou hlavičku se stejnými hodnotami.

Odesílání a čekání na odpověď

Odesílání a příjem ICMP paketů je řešeno pomocí standardních prostředků operačního systému Linux pro práci s počítačovou sítí – *socketů* (výpis 10).

```
icmp_socket = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.
    getprotobyname("icmp"))
icmp_socket.settimeout(0)

icmp_socket.sendto(request, ("10.0.0.1", 0))

start = time()
while (time() - start) <= timeout:
    try:
        reply = icmp_socket.recv(1024)[20:]

        if check_reply(request, reply):
            break
    except BlockingIOError:
        pass

self.icmp_socket.close()
```

Výpis 10: Odesílání a příjem ICMP paketů

Pro práci se sockety je v Pythonu dostupný stejnojmenný modul `socket` [13]. Nejprve je potřeba vytvořit socket pro ICMP provoz. Tomuto socketu je nastaven tzv. neblokovací mód (tzn. pokud se bude ze socketu číst a žádná data nebudou dostupná, nebude se na data čekat, ale místo toho je vyvolána výjimka `BlockingIOError`). Pokud by tento mód socketu nastaven nebyl, ale místo toho by měl socket nastavenu maximální dobu, po kterou má na data čekat, nebylo by možné při dávkovém zasílání paketů účinně a přesně měřit časový limit.

Dalším krokem je odeslání paketu na požadovanou adresu pomocí metody `sendto()`. Argumenty této metody jsou paket a n-tice představující adresu. V případě socketů typu *AF_INET* se jedná o dvojici *(IP adresa, port)*. Protože ale pro ICMP pakety není port relevantní, lze namísto něj použít jakýkoliv platný port.

Následuje smyčka měřící časový limit a přijímající příchozí ICMP pakety. Ty jsou přijímány metodou `recv()`, která má jeden argument a tím je velikost buf-

feru neboli maximální velikost dat, kterou může metoda najednou vrátit. Metoda vrací paket včetně 20bytové hlavičky protokolu IP, ta se proto musí, pokud není potřeba, z paketu odstranit.

Smyčka může být ukončena několika způsoby. Buď byl překročen nastavený časový limit, nebo byl přijat paket, který byl následně zkontrolován a vyhodnocen jako odpověď na odeslanou žádost.

Zpracování odpovědí

Každý přijatý ICMP paket je potřeba zkontrolovat a pokud projde všemi kontrolami, teprve poté se může z časů odeslání žádosti a přijetí odpovědi vypočítat zpoždění v síti.

Nejprve je nutné vypočítat kontrolní součet přijatého paketu. Výpočet je stejný jako výpočet ve výpisu 9, ale s tím rozdílem, že vypočítaný kontrolní součet se nemusí porovnávat se součtem uvedeným v paketu. Pokud se totiž počítá kontrolní součet z paketu, který jej má v sobě obsažený, musí být vypočítaný kontrolní součet roven *0* [16].

Dále je potřeba zkontrolovat typ a kód přijatého paketu. Jak již bylo řečeno výše, pokud nastane někde nějaká chyba, je místo odpovědi zaslán jiný ICMP paket s popisem chyby. Kontrola se provádí pomocí typu a kódu v hlavičce paketu. Seznam všech možných ICMP paketů včetně jejich popisu a číselných kódů je uveden v dokumentu *RFC 792* [15].

A konečně je potřeba zkontrolovat, zda-li se shodují identifikátory, pořadová čísla a data obsažená v žádosti a v odpovědi.

4.3.4 Testování End-to-End spojení

Testování je prováděno pomocí TCP spojení mezi klientem a serverem. Celé testování probíhá následujícím způsobem:

1. Klient vygeneruje náhodné neznaménkové 32bitové číslo (identifikátor) a to posléze odešle serveru a zaznamená si čas odeslání.

2. Při přijetí si server zaznamená čas přijetí paketu. Poté pošle klientovi zpět identifikátor, spolu s časem přijetí a aktuálním časem jakožto časem odeslání.

3. Klient si zaznamená čas přijetí, porovná, zda-li se oba identifikátory shodují, a poté vypočítá čas přenosu ze zaznamených údajů prostým sečtením časů obou přenosů (z klienta na server a zpět). Výhoda výpočtu obou časů až po dokončení přenosů tkví v odstranění případného rozdílu systémových hodin klientského počítače a počítače na kterém je spuštěn server.

Klient i server jsou samostatné třídy využívající pro komunikaci po síti sockety.

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(("0.0.0.0", 10000))
server_socket.listen(0)

client_socket, address = server_socket.accept()
clients_queue.put(client_socket)
```

Výpis 11: TCP server

Na výpisu 11 je ukázka implementace TCP serveru. Ze všeho nejdříve je vytvořen socket typu *SOCK_STREAM*, což je socket pro protokol TCP. Následně je tomuto socketu přiřazena (pomocí metody `bind()`) adresa a port, na kterém má server naslouchat (adresa *0.0.0.0* znamená, že server bude naslouchat na všech IP adresách přiřazených síťovým rozhraním počítače, na kterém server běží). Poté je spuštěno samotné naslouchání (metoda `listen()`).

Server navíc využívá pro obsluhu jednotlivých klientů vlákna. Jednotlivé klientské sockety vrácené metodou `accept()` jsou ukládány do synchronizované fronty (modul `queue`), odkud si je jednotlivá vlákna postupně vyzvedávají a obsluhují.

Co se týká TCP klienta, před samotnou komunikací je nejprve potřeba navázat TCP spojení pomocí metody `connect()` jejímž argumentem je n-tice *(IP adresa, port)*.

Samotná komunikace mezi serverem a klientem poté probíhá pomocí metod `send()` a `recv()`.

4.3.5 Detekce výpadků bezdrátové sítě

Ve třídě `OutageDetector` je implementována vlastní detekce výpadků bezdrátového spojení. Tato třída získává potřebné parametry, které jsou posléze vyhodnocovány.

Aktuální přenosovou rychlost lze získat systémovým voláním podobným jako ve výpisu 6, ovšem s konstantou `SIOCGIWRATE`.

Ze souboru `/proc/net/wireless` jsou získávány další parametry týkající se bezdrátové sítě (kapitola 3.1.3).

Statistika provozu na jednotlivých rozhraních se nachází v adresáři daného rozhraní (např. `/sys/class/net/wlan0`) v podadresáři `statistics`. Údaje získávané z toho adresáře jsou následující: počty odeslaných bytů a paketů (soubory `tx_bytes` a `tx_packets`) a počet bytů a paketů přijatých (`rx_bytes` a `rx_packets`).

Latence sítě je měřena pomocí dvou velikostí ICMP paketů. Krátkých o velikosti několika bytů a dlouhých, jejichž velikost je rovna hodnotě MTU aktuální sítě.

Nakonec je, pokud to není zakázáno nastavením programu, zkontrolováno end-to-end spojení.

Algoritmus detekce výpadků

Základem algoritmu pro detekci výpadků se staly experimenty v kapitole 3. Konkrétní podoba algoritmu však byla vytvořena až podrobným testováním na zmíněných sítích. Celý algoritmus je vyjádřen pomocí pseudokódu v příloze B.

Hlavním měřítkem pro detekci výpadku je latence sítě a úspěšnost odeslaných ICMP paketů. Aby byla tato hodnota vypovídající, je v závislosti na aktuálních podmínkách (úroveň signálu, apod.) měněn počet těchto paketů. Algoritmus se také přizpůsobuje aktuálnímu provozu na daném síťovém rozhraní.

4.3.6 Bezpečnost

Vzhledem k tomu, že program musí být spuštěn se superuživatelskými právy (zejména kvůli zasílání ICMP paketů), je implementováno několik bezpečnostních mechanismů.

Prvním je kontrola vlastnictví a přístupových práv ke konfiguračnímu souboru a ke všem souborům se zdrojovými kódy programu. Všechny zmiňované soubory musí být ve vlastnictví uživatele *root* a skupiny *root*. Co se týká kontroly práv, mohou být jakákoliv, ale právo zápisu může mít pouze vlastník a skupina.

Další bezpečnostní mechanismus spočívá v tom, že se superuživatelskými právy se provádějí pouze nezbytné části programu. Zbytek programu běží s právy normálního uživatele.

Proces spuštěný uživatelem *root* může dočasně snížit a posléze zase povýšit zpět svá oprávnění nastavením tzv. *EUID* a *EGID*[4], která určují oprávnění daného procesu.

V Linuxovém systému existuje standardní uživatel *nobody* (a skupina *nobody*), který má pouze základní uživatelská práva. Stačí tedy zjistit *UID* a *GID* tohoto uživatele a skupiny (ze souborů `/etc/passwd` a `/etc/group`, případně v Pythonu pomocí modulu `pwd` [12]) a nastavit tyto hodnoty procesu.

[4]Nebo svá oprávnění snížit trvale nastavením *UID* a *GID*.

5 Závěr

Závěry získané pomocí uvedených experimentů se shodují s teoretickými předpoklady uvedenými v úvodu práce.

Výsledný program byl řádně otestován na Wi-Fi sítích uvedených v kapitole 3. Testování se zaměřovalo na to, zda výpadky byly detekovány pouze v opravdu nezbytných případech, kdy se síť stávala z uživatelského hlediska nepoužitelnou. Při testech bylo zjištěno, že několik okamžiků (v řádech zlomků sekundy) po detekci výpadků tímto programem následovalo odpojení od sítě.

Výsledkem této práce je tedy plně funkční program poskytující spolehlivou detekci výpadků a celkové nedostupnosti Wi-Fi sítě.

Další vývoj tohoto projektu by mohl spočívat ve využití tzv. *NetLink socketů*. Jedná se o modernější způsob komunikace uživatelských procesů s jádrem systému pomocí socketů podobných těm síťovým. Ale vzhledem k relativně komplikovanému způsobu zpracovávání zpráv posílaných pomocí těchto socketů a velkému počtu maker jazyka C/C++ potřebných pro tuto komunikaci, by se výsledný program v jazyce Python stával příliš složitým a nepřehledným.

Další vývoj tohoto projektu by mohl být také např. v napojení tohoto programu na – v dnešních distribucích OS Linux tolik oblíbený – program *NetworkManager*. Jedná se o správce síťových připojení v grafickém prostředí OS Linux. Při experimentech, kde byl tento správce částečně také využíván, byly pozorovány problémy, které měl při výpadcích bezdrátového spojení jež byly detekovány s velkým zpožděním.

6 Reference

[1] GAST, M. *802.11 wireless networks: the definitive guide*. 2nd edition. O'Reilly, 2005. ISBN 978-0-596-10052-0.

[2] MADHOW, U. *Fundamentals of digital communication*. Cambridge University Press, 2008. ISBN 978-0-521-87414-4.

[3] SMITH, R. *Advanced Linux Networking*. Addison-Wesley, 2002. ISBN 978-0-201-77423-8.

[4] WIKIMEDIA COMMONS. *Interference of sine waves*. Dostupné z: `http://commons.wikimedia.org/wiki/File:Interference_of_sine_waves.JPG`

[5] WIKIMEDIA COMMONS. *Wifi hidden station problem*. Dostupné z: `http://en.wikipedia.org/wiki/File:Wifi_hidden_station_problem.svg`

[6] *The Linux Documentation Project* [online]. [cit. 2013-04-15]. Dostupné z: `http://www.tldp.org`

[7] PYTHON SOFTWARE FOUNDATION. *Python Programming Language: Official Website* [online]. [cit. 2013-04-24]. Dostupné z: `http://www.python.org`

[8] Argparse. *Python documentation* [online]. Dostupné z: `http://docs.python.org/3/library/argparse.html`

[9] Array. *Python documentation* [online]. Dostupné z: `http://docs.python.org/3/library/array.html`

[10] Configparser. *Python documentation* [online]. Dostupné z: `http://docs.python.org/3/library/configparser.html`

[11] Fcntl. *Python documentation* [online]. Dostupné z: `http://docs.python.org/3/library/fcntl.html`

[12] Pwd. *Python documentation* [online]. Dostupné z:
`http://docs.python.org/3/library/pwd.html`

[13] Socket. *Python documentation* [online]. Dostupné z:
`http://docs.python.org/3/library/socket.html`

[14] Struct. *Python documentation* [online]. Dostupné z:
`http://docs.python.org/3/library/struct.html`

[15] RFC 792. *Internet control message protocol*. 1981. Dostupné z:
`http://tools.ietf.org/html/rfc792`

[16] RFC 1071. *Computing the internet checksum*. 1988. Dostupné z:
`http://tools.ietf.org/html/rfc1071`

A Příloha na CD

Obsah CD

Přiložené CD obsahuje ZIP soubor s kompletní implementací programu pro detekci výpadků v prostředí Wi-Fi sítí.

B Algoritmus detekce výpadků

```
while není detekován výpadek do
    if existují chybějící beacon rámce then
        detekce výpadku
    end if

    if chybných paketů je víc než 75 % odpovídajícího provozu then
        detekce výpadku
    end if

    if úroveň signálu klesne pod -80 dBm then
        zdvojnásobení počtu krátkých ICMP paketů
    end if

    if celkový provoz je větší než polovina přenosové rychlosti then
        polovina počtu krátkých ICMP paketů
        zdvojnásobení časového limitu pro ICMP pakety
    end if
    if celkový provoz je větší než čtvrtina přenosové rychlosti then
        zákaz zasílání dlouhých ICMP paketů
    end if

    if úspěšnost dlouhých ICMP paketů je 0 % then
        zdvojnásobení počtu krátkých ICMP paketů
    end if
    if úspěšnost krátkých ICMP paketů je menší než 25 % then
        detekce výpadku
    end if

    if není zakázáno end-to-end spojení then
        if end-to-end spojení selhalo then
            detekce výpadku
        end if
    end if
end while
```

C Nápověda k programu

C.1 Spouštění programu

Program vyžaduje interpret jazyka *Python* a to ve verzi minimálně *3.3*. Poté lze spustit pomocí souboru `main.py` dvěma způsoby:

- **`./main.py`** – Vyžaduje aby soubor `./main.py` měl práva ke spouštění.
- **`python main.py`**

C.2 Parametry příkazové řádky

`-h/--help`
: Zobrazí nápovědu k programu se všemi možnými parametry.

`-v/--verbose`
: Vypisuje informační zprávy o vykonávaných akcích a zjištěných informacích.

`-c FILE/--conf FILE`
: Cesta ke konfiguračnímu souboru (pokud není zadána, hledá se konfigurační soubor v adresáři, kde je program uložen).

`-s/--server`
: Program se spustí v režimu serveru pro testování end-to-end spojení.

`--no-end-to-end`
: Parametr určený pouze pro režim klienta. Nebude se testovat end-to-end spojení (například pokud není dostupný žádný server).

C.3 Konfigurační soubor

Konfigurační soubor se skládá ze dvou sekcí (`[client]` a `[server]`) pro režim klienta a serveru. Každá sekce obsahuje vlastní možnosti ve tvaru `název = hodnota`. Pokud některá z možností chybí (nebo pokud není nalezen konfigurační soubor), použije se místo ní výchozí hodnota.

[client]

disconnect_action

Příkaz operačního systému, skript nebo program, který se spustí, je-li detekován výpadek.

Výchozí hodnota: *Není*

infinite

Určuje zda-li se program po detekování výpadku ukončí nebo bude pokračovat znova od začátku.

Výchozí hodnota: *1*

interface

Určuje síťové rozhraní, na kterém má program detekovat výpadky. Pokud je zadaná hodnota „auto“, program automaticky prochází všechna dostupná bezdrátová síťová rozhraní a výpadky kontroluje na prvním, které se připojí k bezdrátové síti.

Výchozí hodnota: *auto*

server_ip

IP adresa serveru, který se má použít pro kontrolu end-to-end spojení. Pokud není IP adresa zadána, automaticky se zakáže kontrola end-to-end spojení.

Výchozí hodnota: *Není*

server_port

Port na kterém server pro kontrolu end-to-end spojení naslouchá.

Výchozí hodnota: *10000*

[server]

listening_ip

IP adresa, na které má server naslouchat. Je-li adresa ve tvaru „0.0.0.0“, server bude poslouchat na všech IP adresách přiřazených počítači, na kterém je spuštěn.

Výchozí hodnota: *0.0.0.0*

`listening_port`

Port na kterém má server naslouchat.

Výchozí hodnota: *10000*

`threads`

Maximální počet vláken spouštěných pro komunikaci s klienty.

Výchozí hodnota: *4*